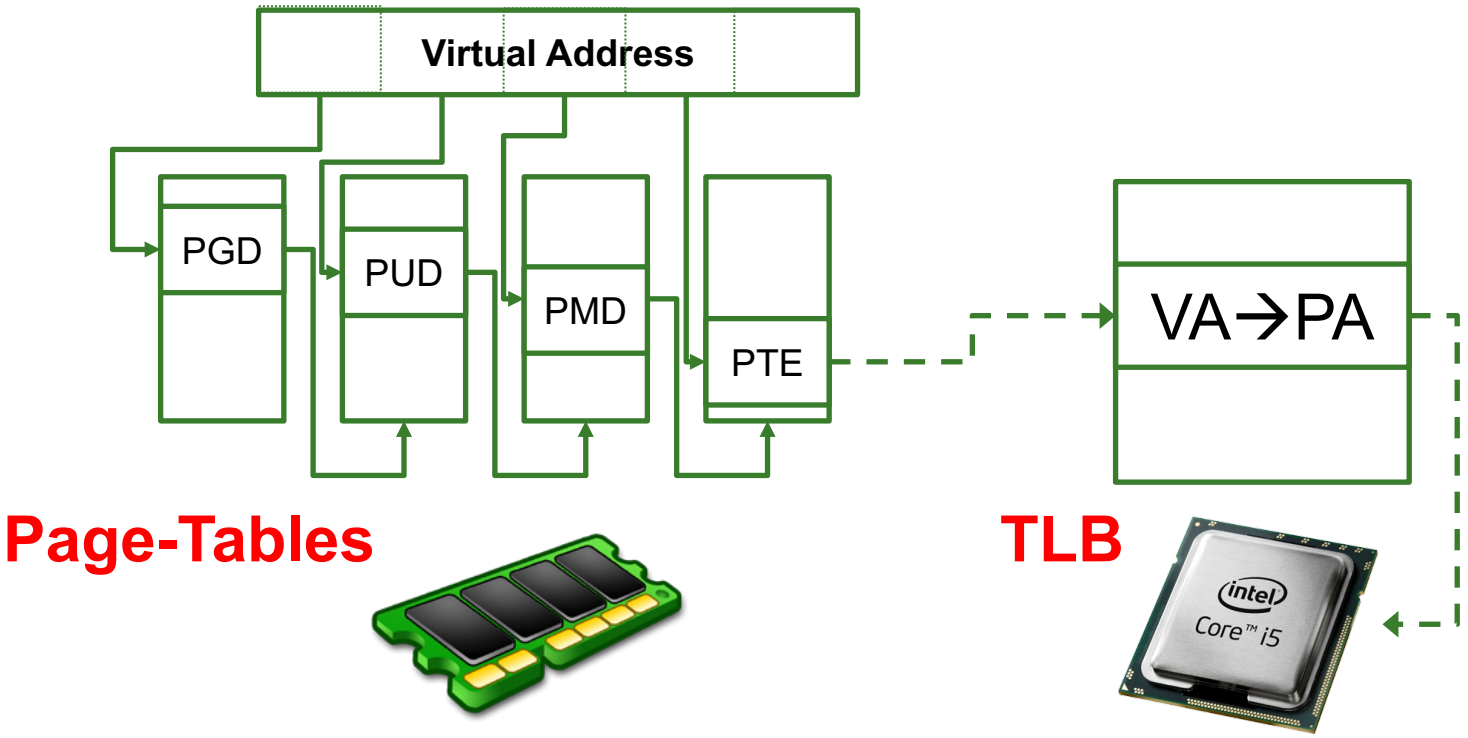
Optimizing the TLB Shootdown Algorithm with Page Access Tracking

Nadav Amit – VMware Research Group

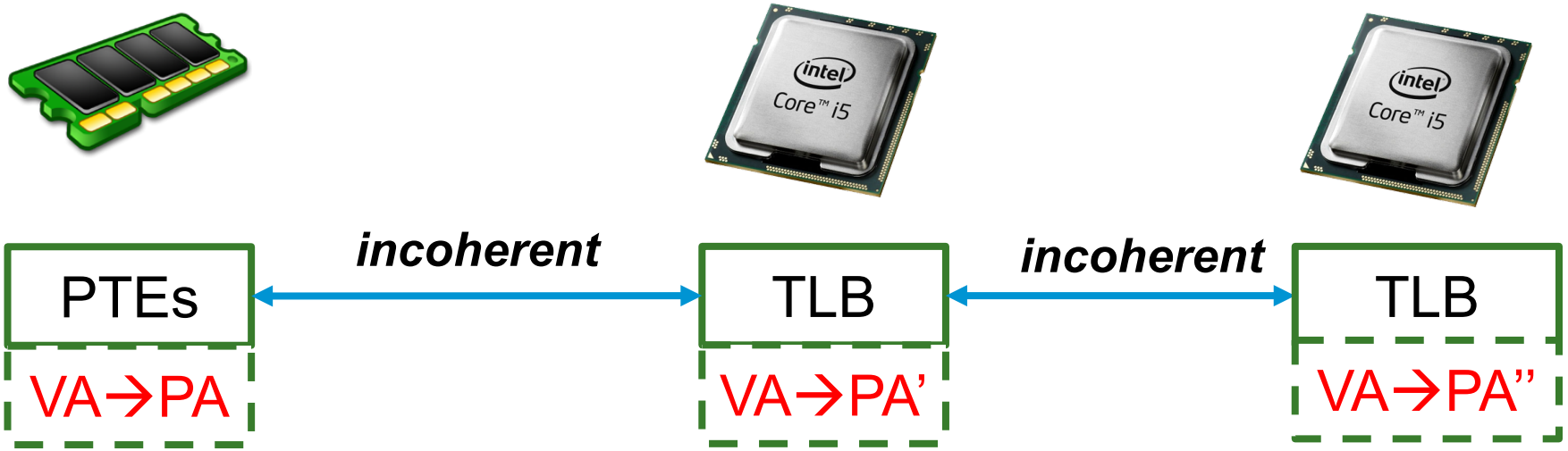
vmware®

Translation Lookaside Buffer (TLB)



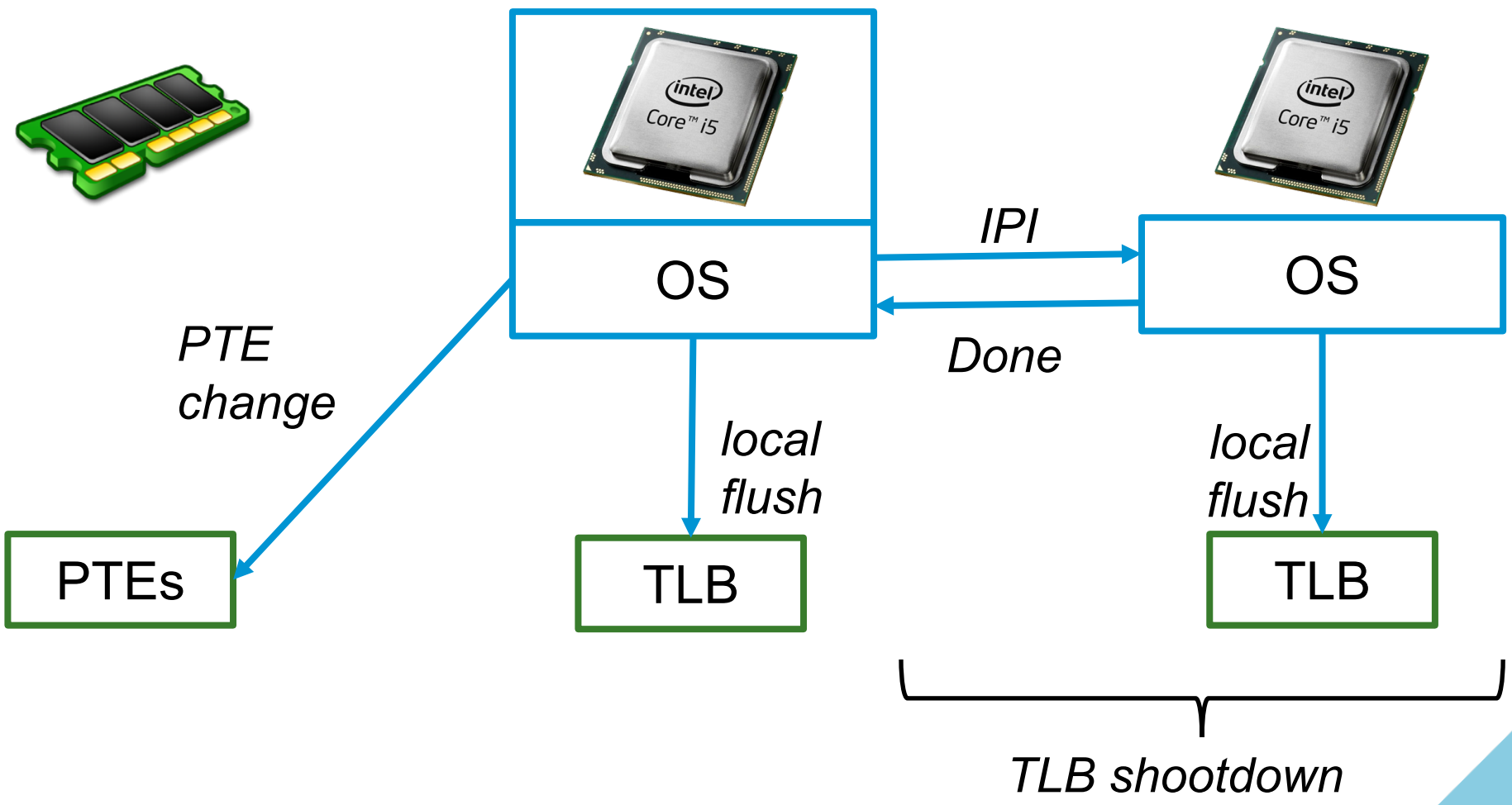
- TLB = cache for virtual to physical address translations

TLB Coherency



- Hardware does not maintain TLBs coherent
- The problem is left for software (OS)

Local TLB Flushes and Remote TLB Shootdowns



When do TLB Flushes Occur?

- Application initiated
  - munmap()
  - Copy-on-write
  - msync()
  - mprotect()
  - madvise()
  - migrate_pages()
- OS initiated
  - NUMA migrations
  - Memory compaction
  - Memory deduplication
  - Memory reclamation
  - Memory balloon
  - Background dirty-pages flush

- Faster storage → TLB overheads are more apparent

Existing Solutions

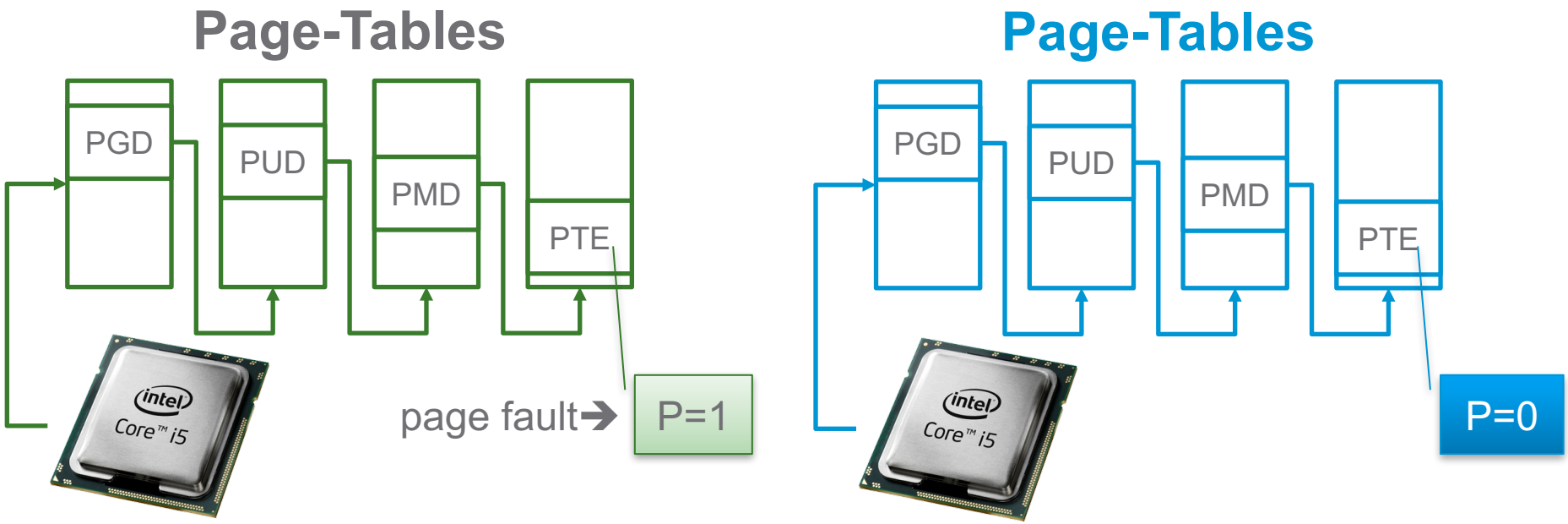
- Hardware [Teller'90, Villavieja'11, Li'13]

- Software (commodity OSes)
  - Batching [Uhlig'05]
  - Limit flushes to cores that use the address-space
  - Trade-off between full and individual PTE flushes

- Software (academic)
  - Explicit software control [Boyd-Wickizer'10, Tene'11]
  - Replicated paging hierarchy [Clements'13, Gerofi'13]

Replicated Paging Hierarchy

[Clements'13, Gerofi'13]



- Page-fault on each CPU that accesses a PTE
- Memory overheads
- Runtime overheads: managing multiple tables

Insight: Use PTE Access-Bit

- Set by hardware, cleared by software

- Used for OS **memory reclamation decisions**
  - Set when a page is **accessed**
  - "These flags are provided ... to manage the transfer of pages … into and out of physical memory." (Intel SDM)

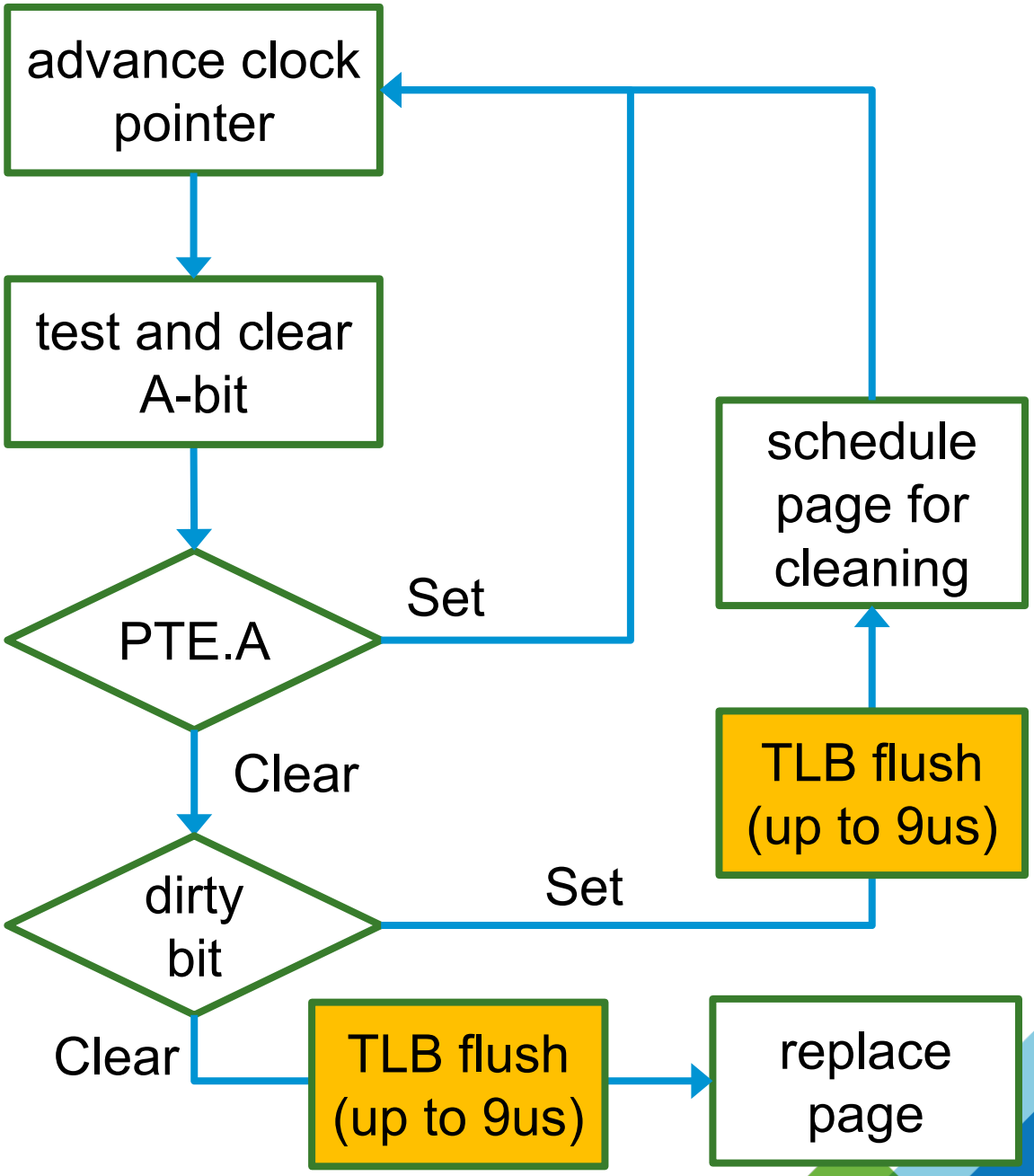
- Insight: **can be used for TLB invalidation decisions**
  - Set when a PTE is **cached**
  - "Whenever the processor uses a PTE as part of address translation, it sets the accessed flag…" (Intel SDM)

Our System

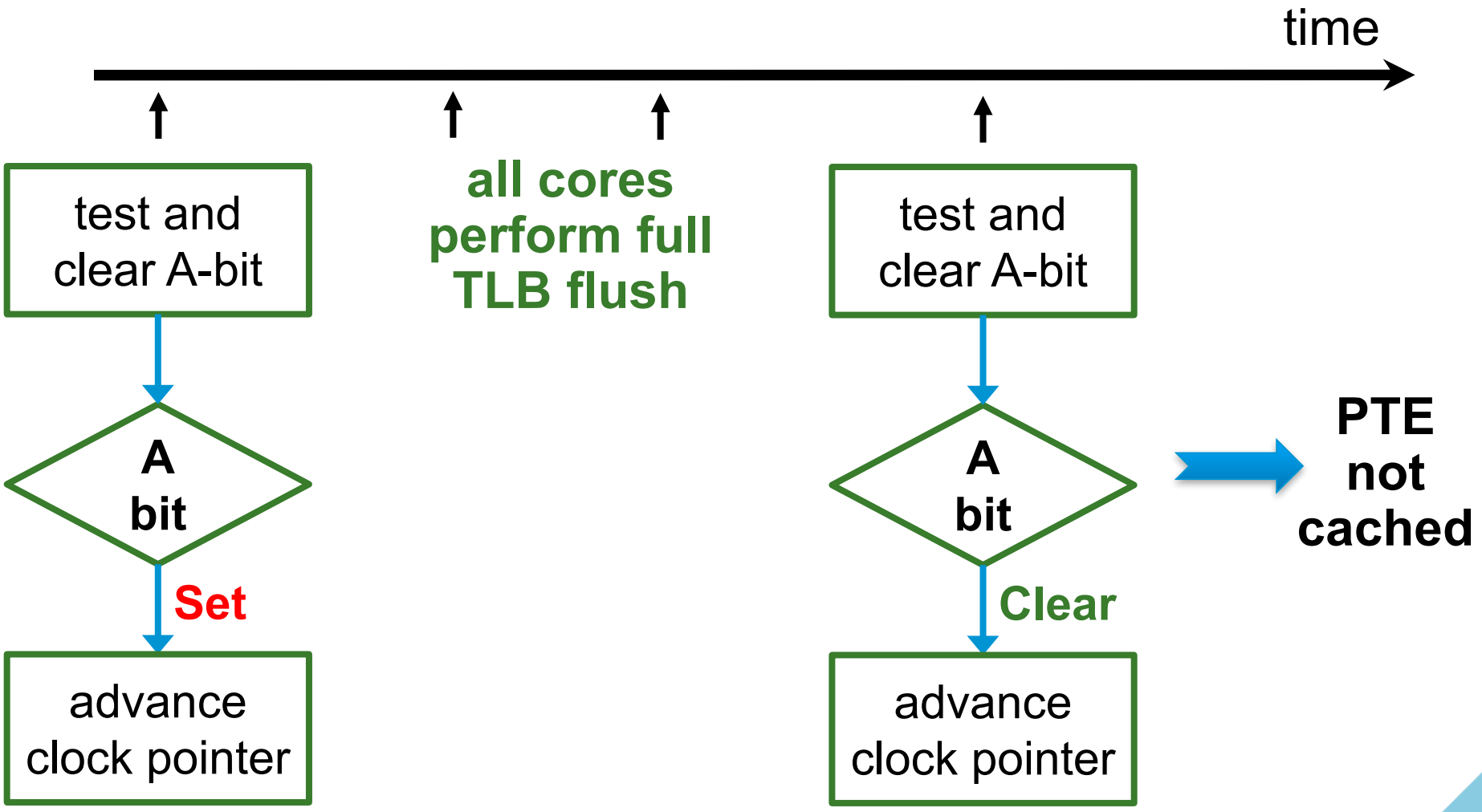
- Flush decisions based on PTE access-bit
- Software solution (x86)
  - Exploiting the full potential requires simple hardware changes
- Prevent common unnecessary TLB shootdowns
  - Long-lived idle mappings
  - Short-lived private mappings
- Some false positives
  - Unnecessary flushes

Long-Lived Idle Mappings

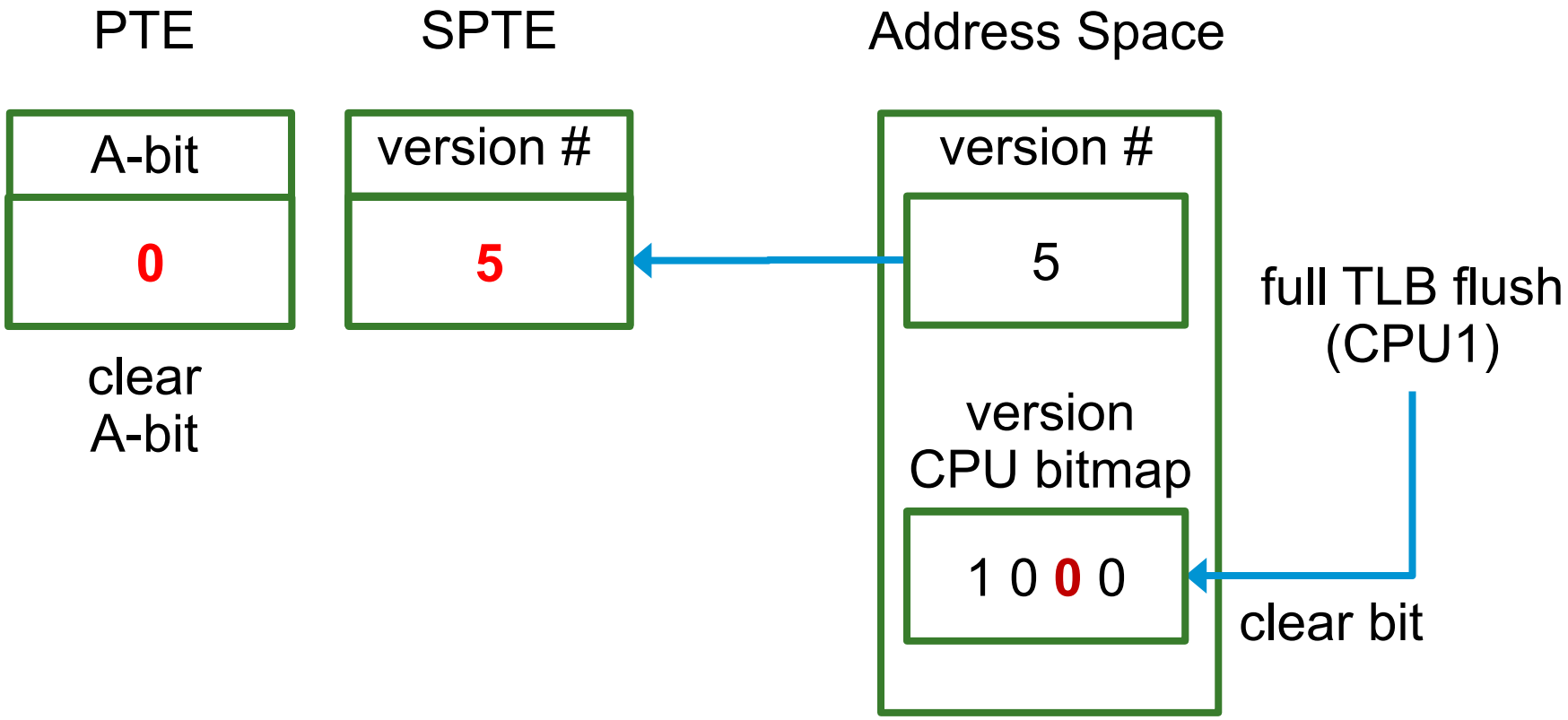
CLOCK Algorithm
[Carr and Henessy '81]



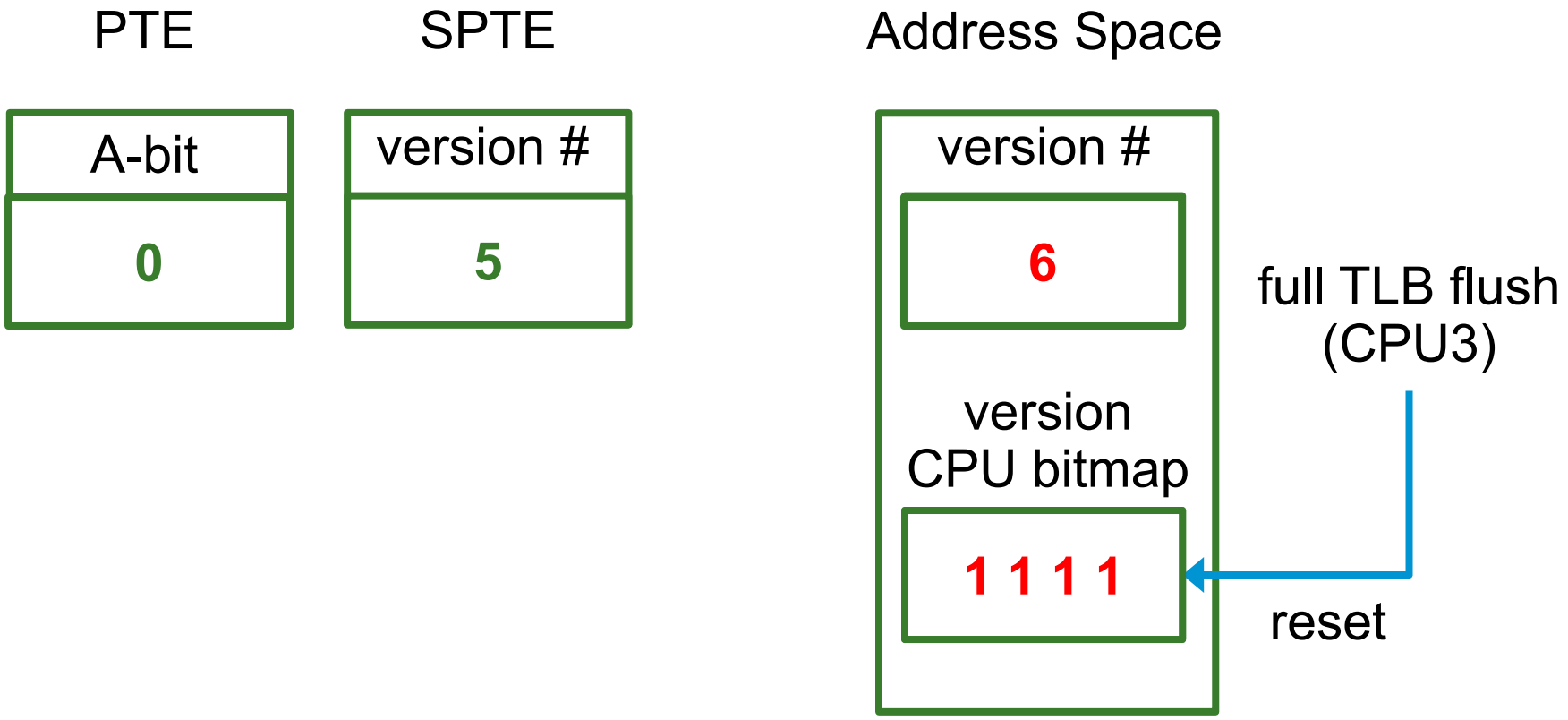
Avoiding Flush of Long Lived Idle PTEs



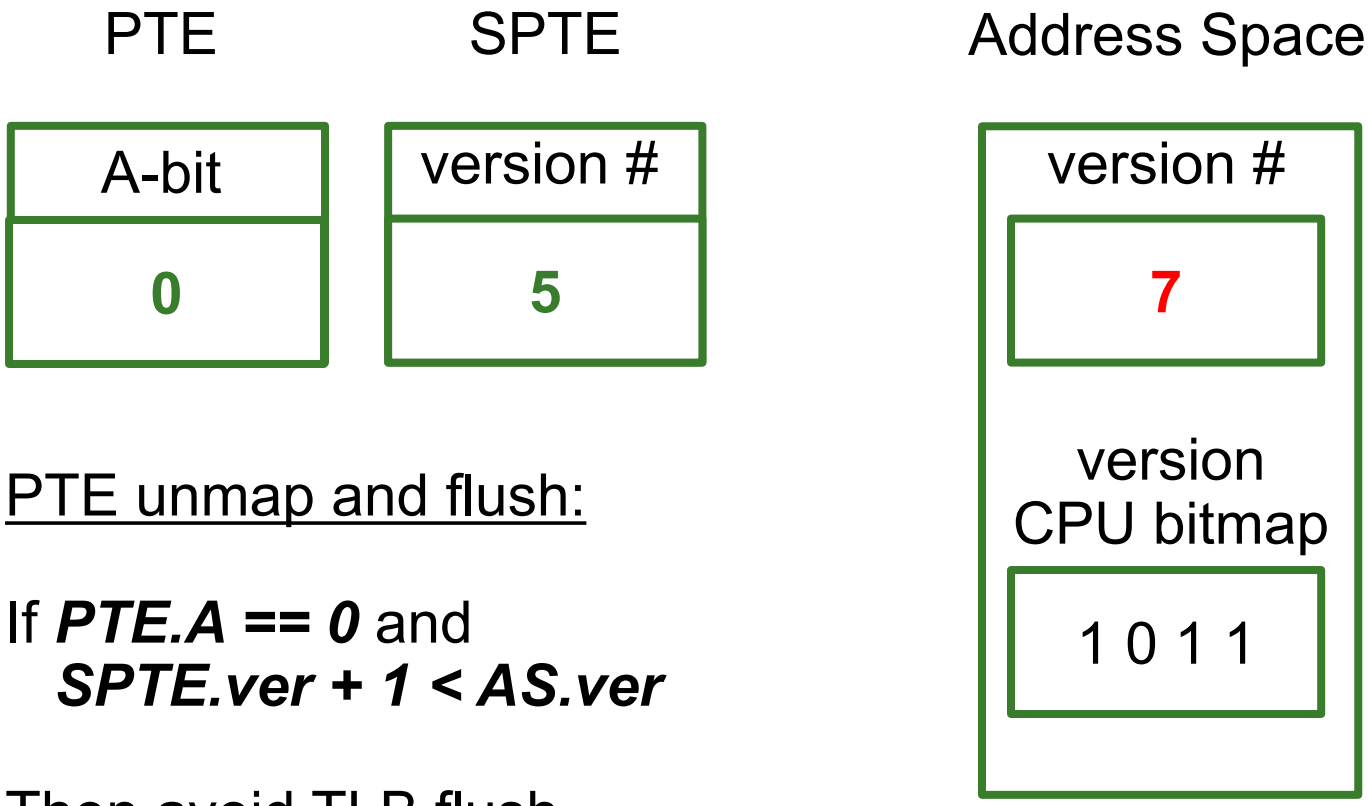
TLB Version Tracking (1)



TLB Version Tracking (2)



TLB Version Tracking (3)

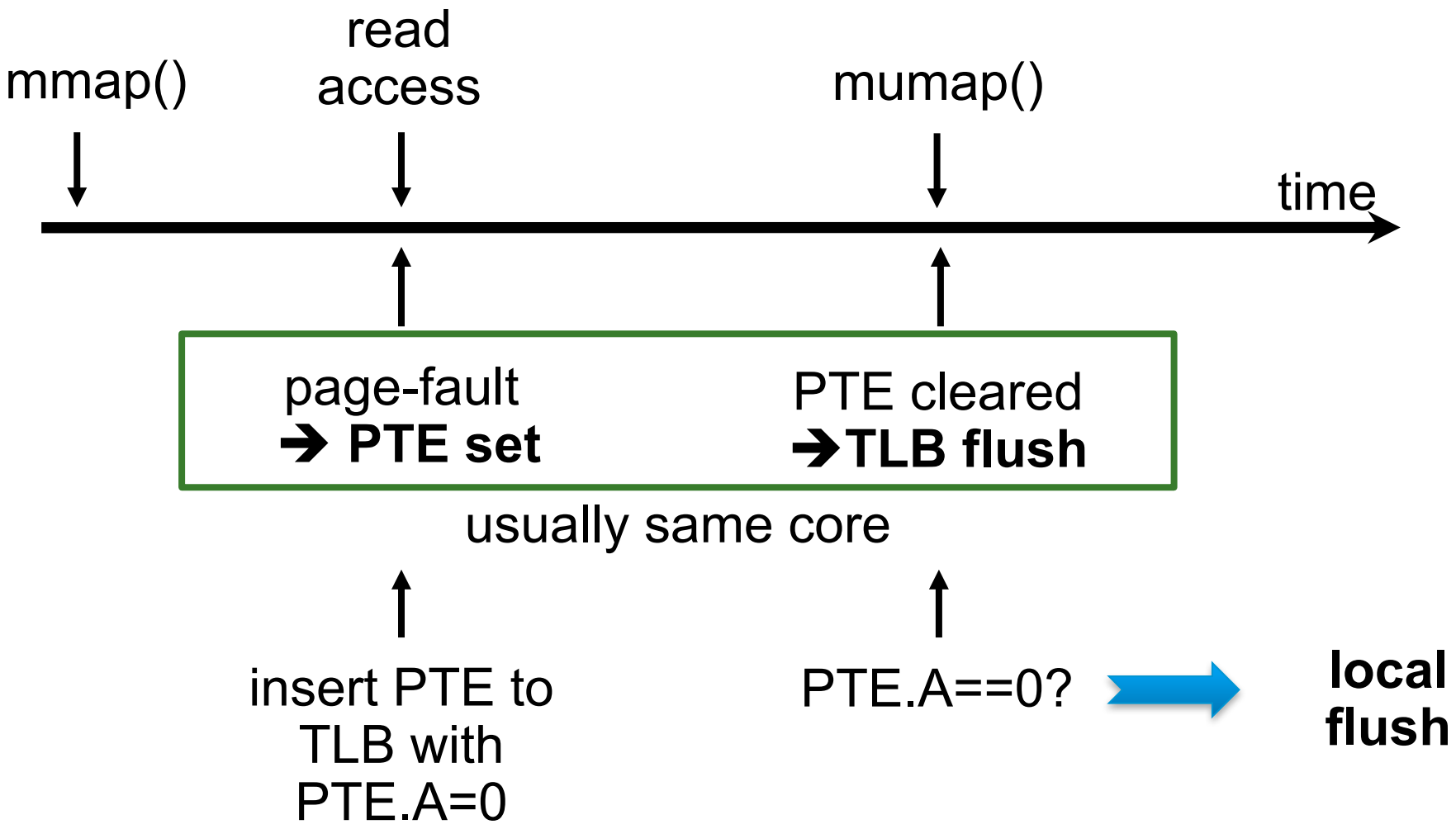


PTE unmap and flush:

If ***PTE.A == 0*** and
SPTE.ver + 1 < AS.ver

Then avoid TLB flush

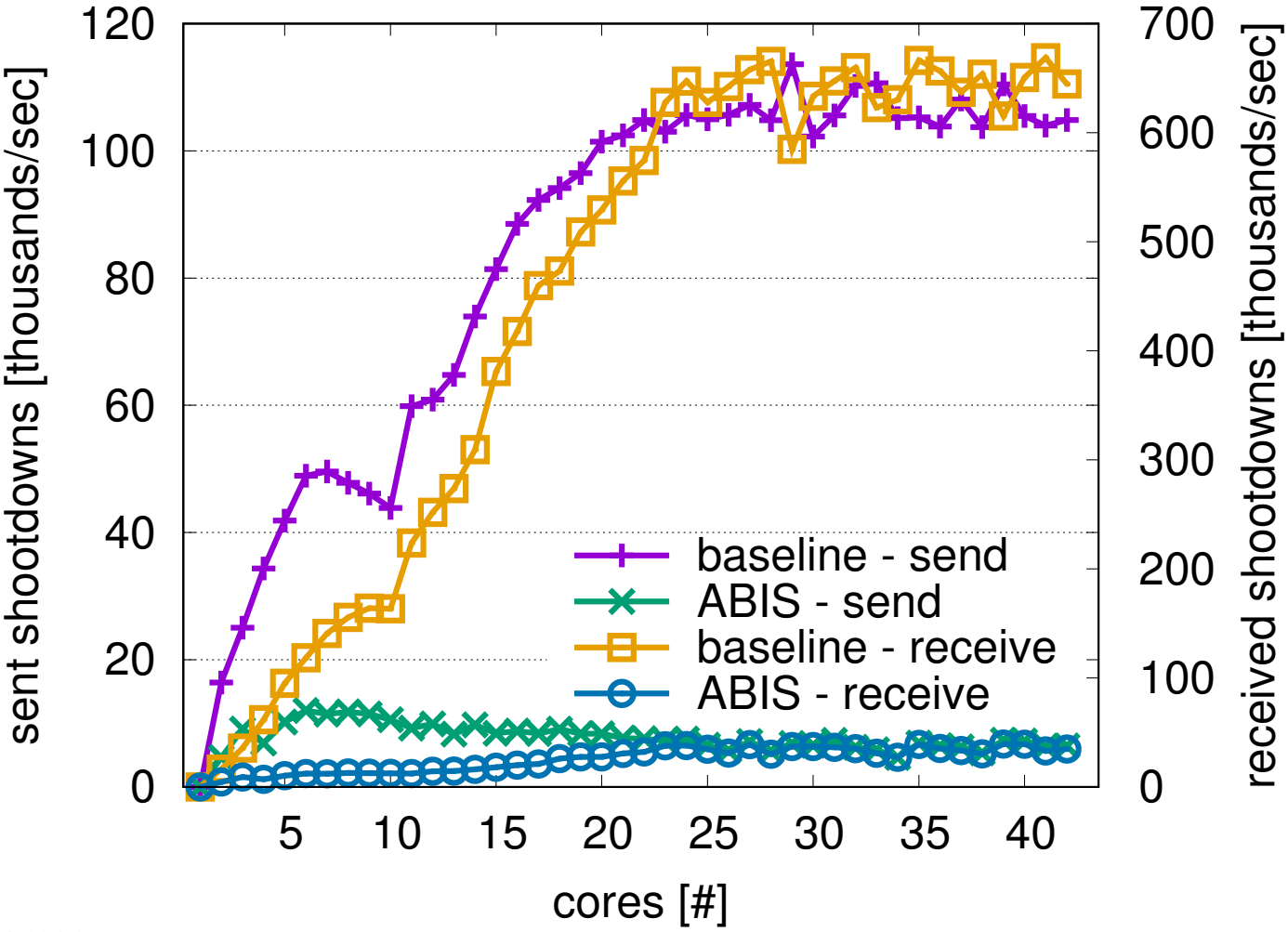
Short Lived Private Mappings



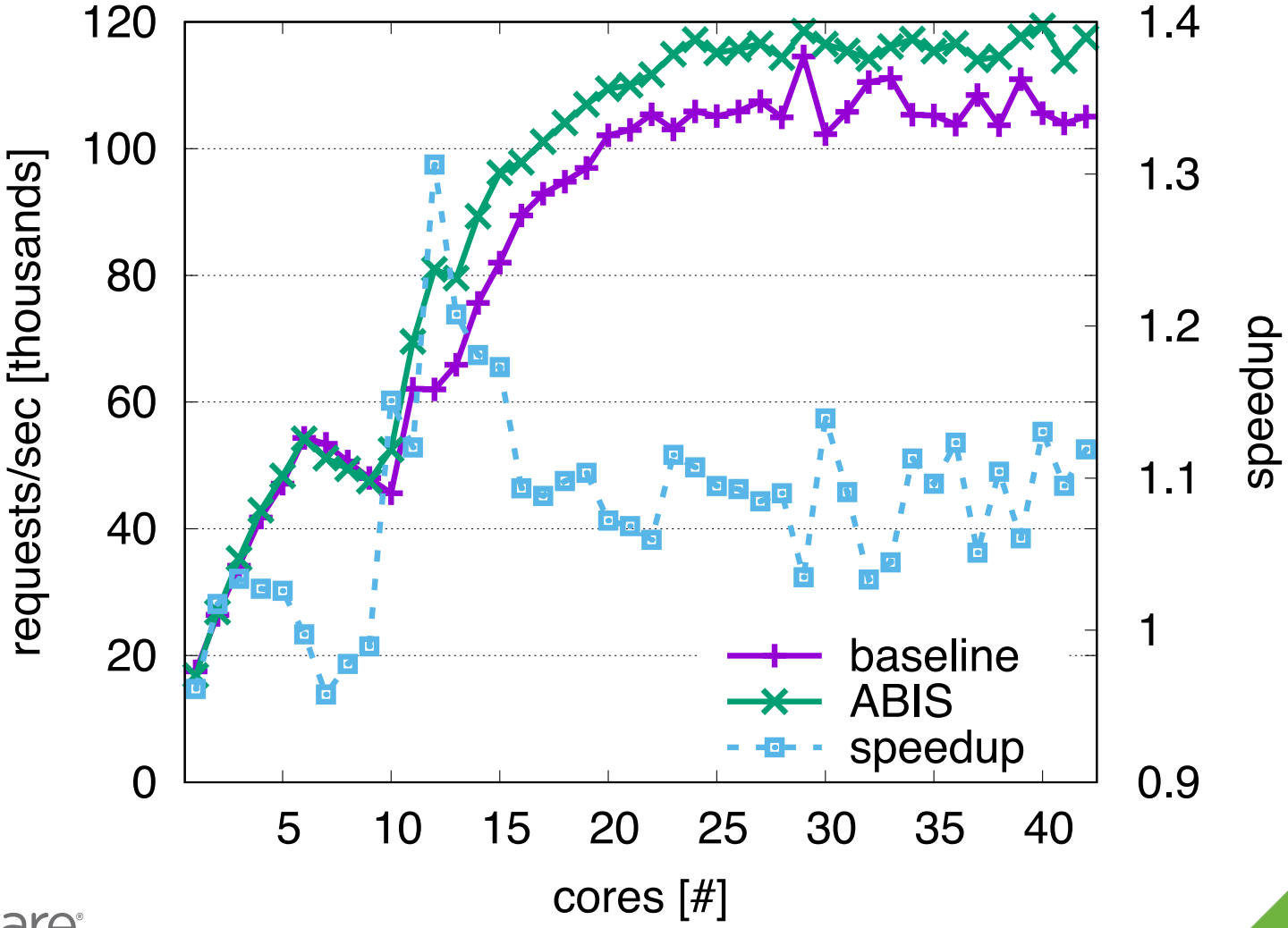
Evaluations

- Prototype based on Linux 4.5
- Baseline configured to avoid shootdown cost
  - Linux version that uses TLB flushes batching
  - Using efficient multicast IPI delivery
- 48-cores, 2-socket server
- Our system denoted as ABIS:
  Access-Based Invalidation System

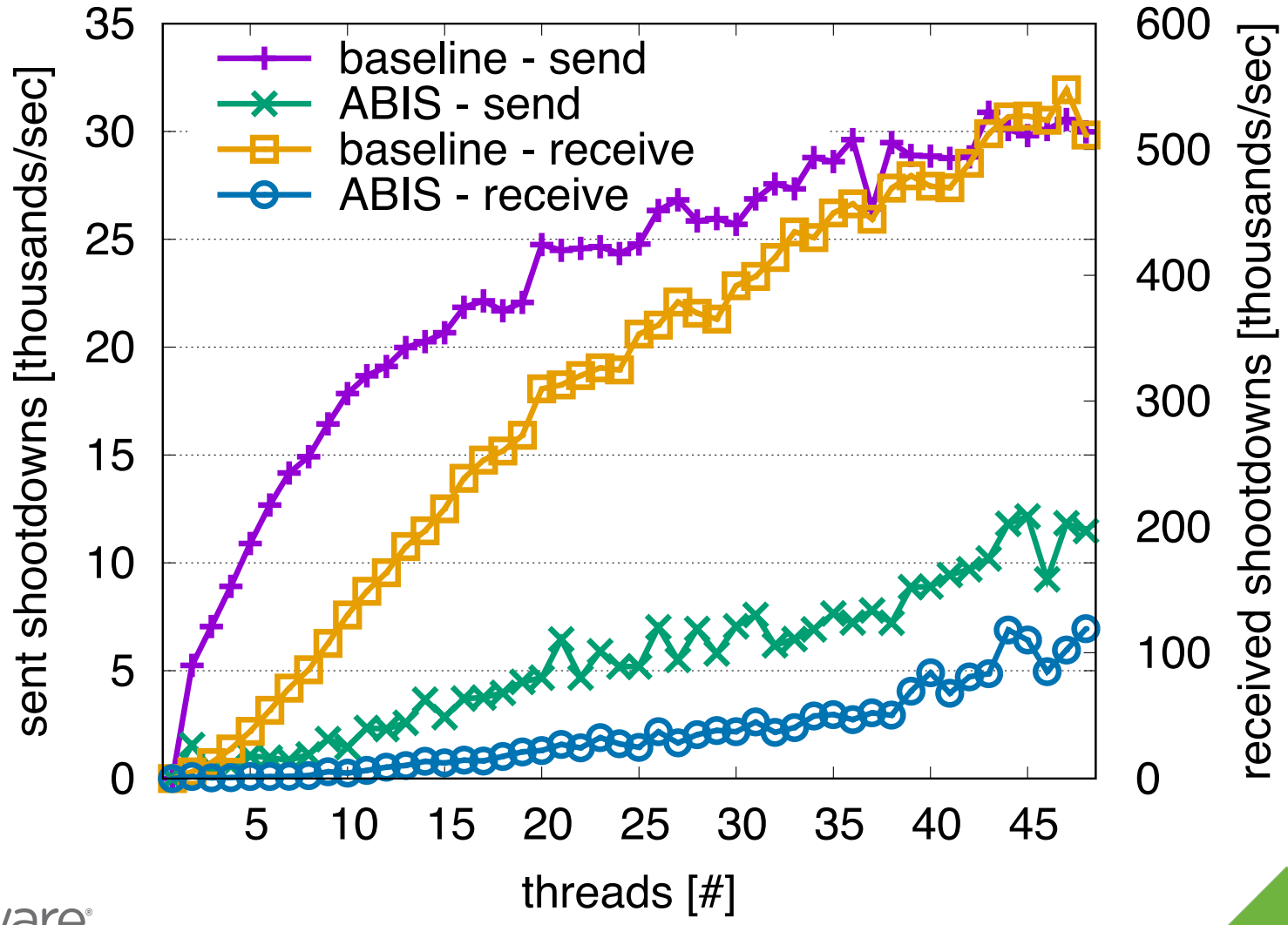
Apache TLB Shootdowns
(Short-Lived Private Mappings)



Apache Performance

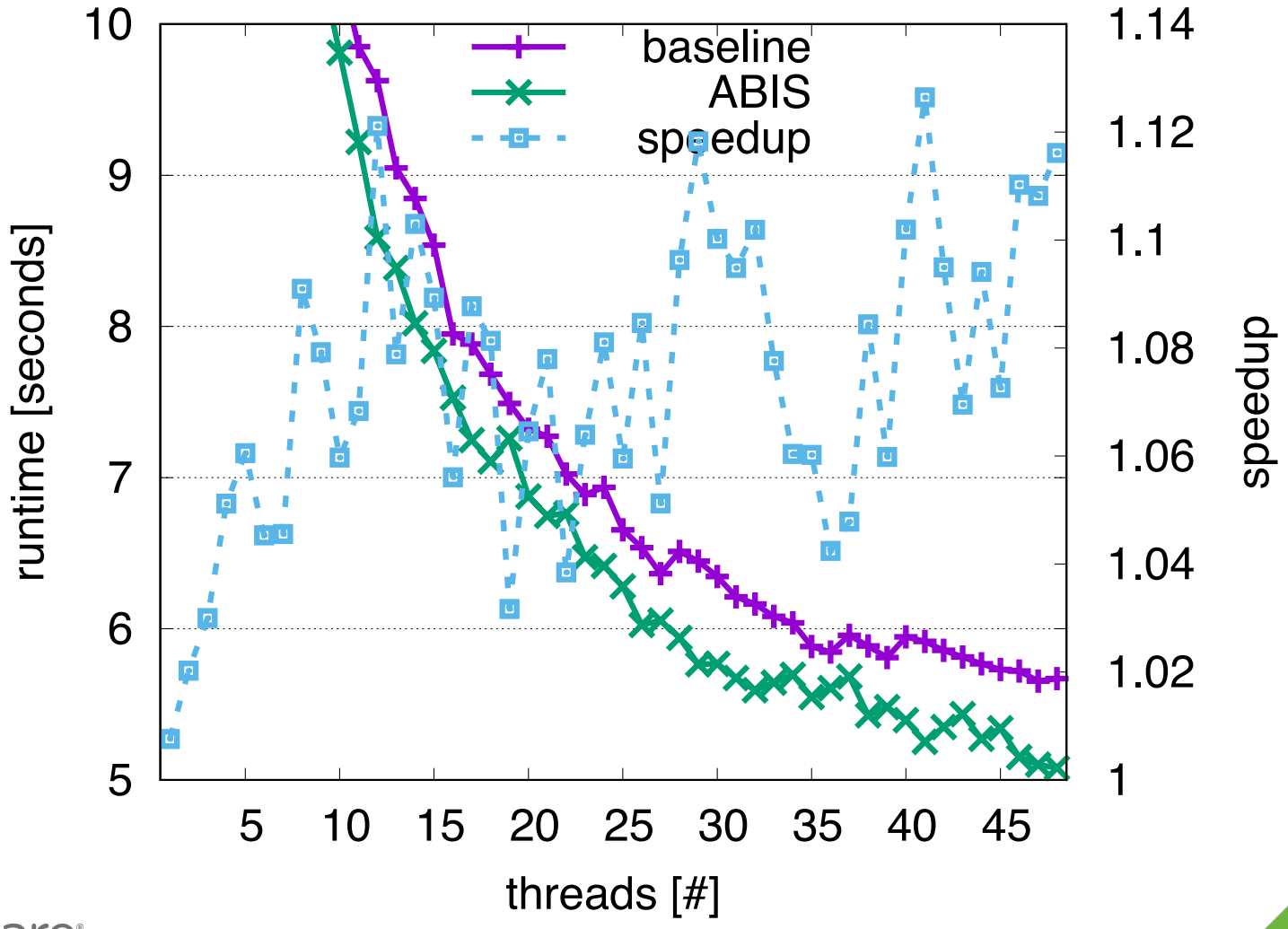


PBZIP2 – TLB shootdowns
(Long-Lived Idle Mappings)

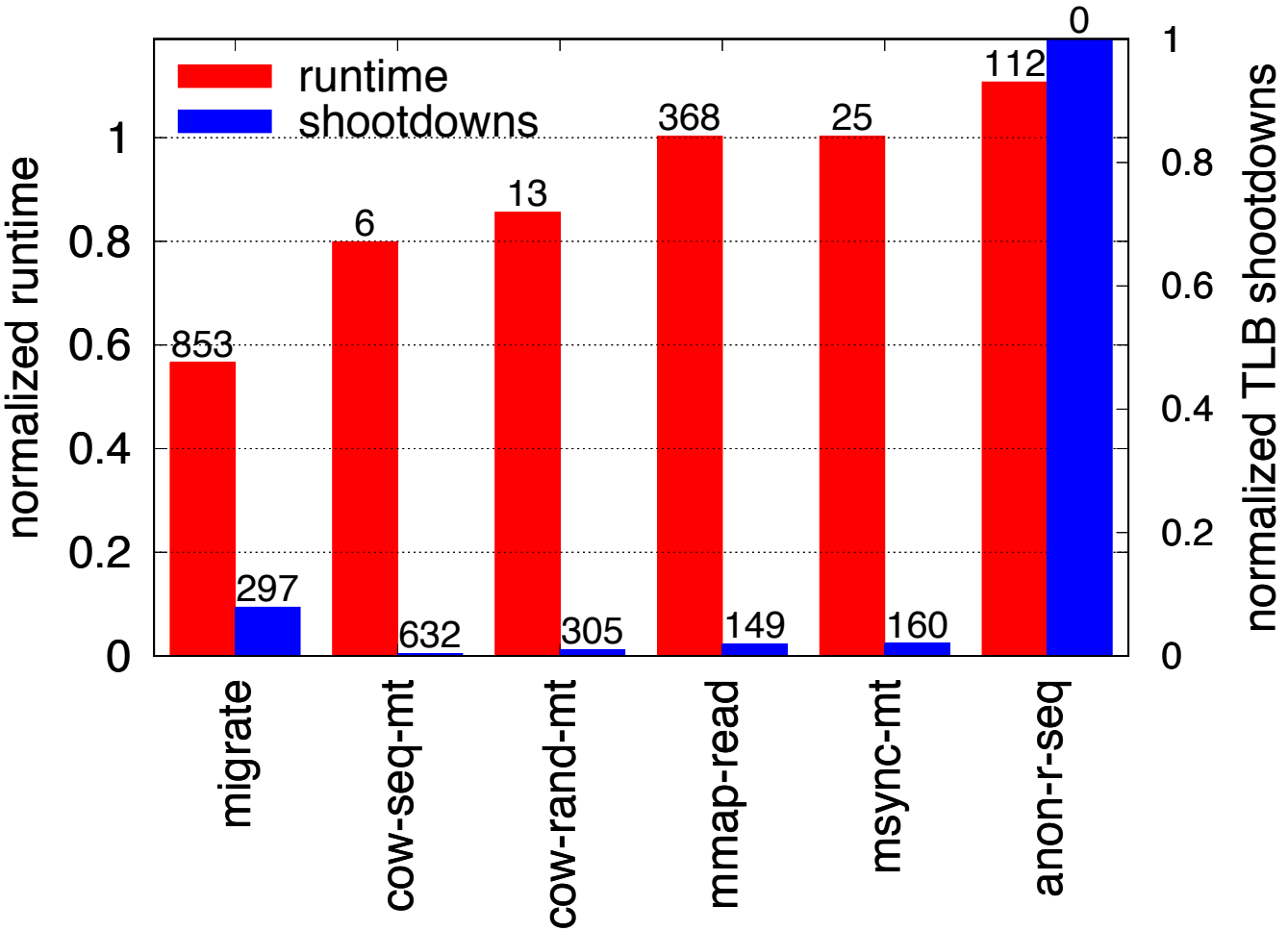


vmware

PBZIP2 Performance



Microbenchmarks: VMScale



Conclusions

- Access-bit tracking can often prevent most TLB shootdowns:
  - Long-lived idle PTEs
  - Short-lived private PTEs
- Exploit memory coherency to check if TLB is cached
- CPUs should allow more control over the TLB
  - Insertion of PTEs directly to the TLB